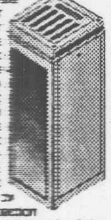
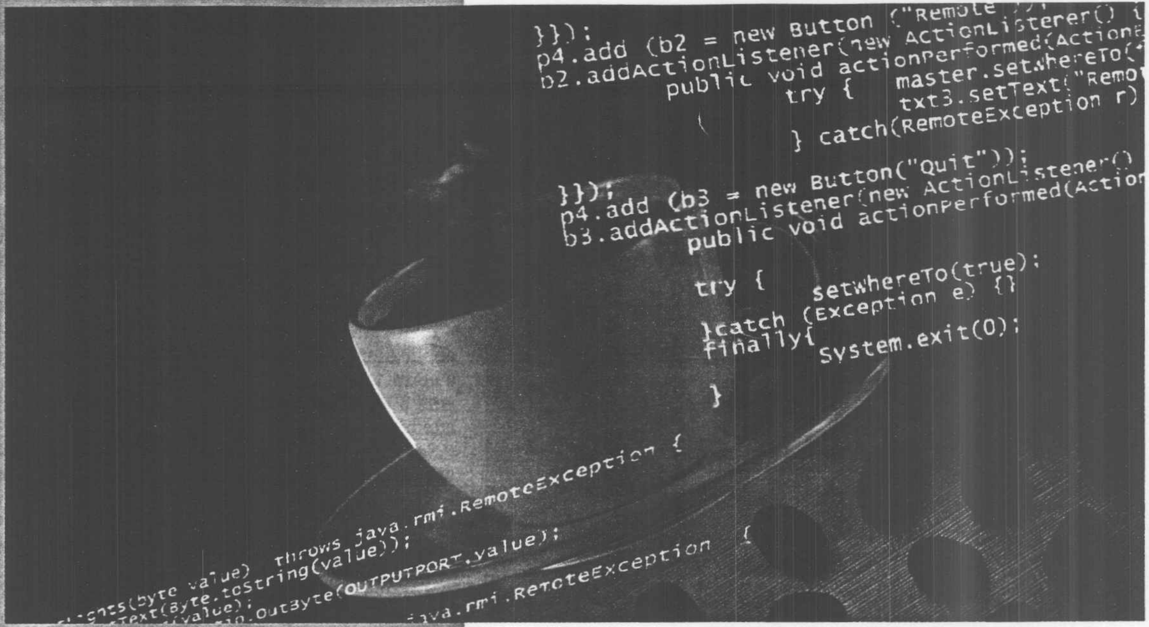




}});
p4.add (b2 = new Button ("Remote"));
b2.addActionListener(new ActionListerer() {
public void actionPerformed(ActionE
try { master.setWhereTo(
txt3.setText("Remo
} catch(RemoteException r)
}});
p4.add (b3 = new Button("Quit"));
b3.addActionListener(new ActionListener()
public void actionPerformed(Action
try { setWhereTo(true);
}catch (Exception e) {}
finally{ System.exit(0);
}
...nts(byte value) throws java.rmi.RemoteException {
...Text(Byte.toString(value));
...(value);
...OutByte(OUTPUTPORT.value);
java.rmi.RemoteException {

Tini Java

Java expert Les Hughes has been experimenting with a tiny controller designed for Internet connection. Although very low cost, this controller is a complete computer with Internet, network and serial i/o capabilities, giving it huge potential for remote i/o and telemetry applications.

Much noise has been made concerning Java. Although originally conceived as a technology capable of powering embedded systems, it is only recently that Java devices have appeared on the market. One of the most interesting of these is the Tiny InterNet Interface – or TINI – from Dallas Semiconductors.

TINI executes a Java Virtual Machine, which in turn executes Java 'bytecodes' – in a similar manner to any other Java platform. However, although TINI runs Java, it is not a hardware implementation of a VM. Instead, the current TINI hardware is based on the Dallas 80C390 micro controller.

The TINI VM, operating system and various user programs are loaded into flash memory. This enables simple updates, which are freely downloadable from the project web site – www.ibutton.com/TINI.

The current TINI board incorporates a plethora of external interfaces: 10baseT ethernet, Dallas one-wire, I^{2}C and RS232. Besides the processor, the TINI board also contains 512Kbyte of flash memory, 512Kbyte of NVRAM and an RTC. Processor address, data and control busses are brought out to the edge of the board for custom expansion – e.g. memory expansion or custom i/o. That's quite a few features for a device that is no bigger than a normal PC SIMM and only costs $50.00!

At the time of writing, TINI firmware was nearing beta 3. The processor board has been slightly redesigned to fit a more standard 72-pin SIMM socket rather than the original 68-pin connector. Although TINI is still beta for both hardware and firmware, recent releases have enhanced performance, stability and features.

Of course, as with any beta product, revisions often occur and APIs and specifications change, based on

Les is a Technical Architect with Rubus.com – a fast growing e-Business Consultancy, currently working with a number of dot-com clients.

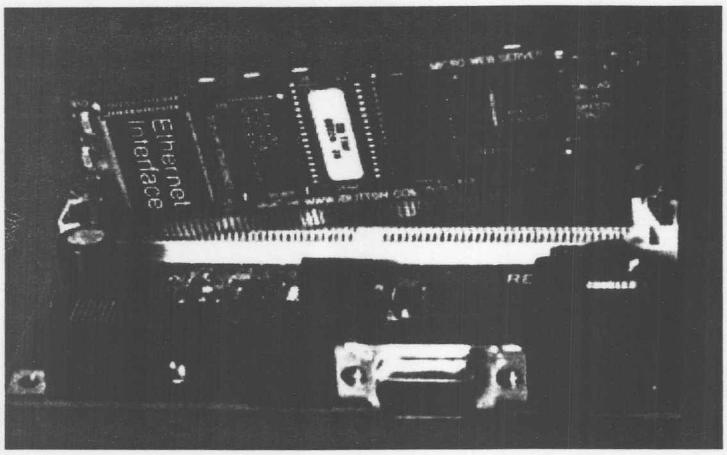
Free software

All of the software required to develop applications for TINI is available free of charge from various Internet sites. Installing and configuring your environment in order to get TINI up and running consists of a number of tasks. Guillaume Fournier's excellent guide at

http://www3.sympatico.ca/guillaume.fournier/

describes in detail the process that you should follow in order to be able to boot your TINI.

user feedback. In a similar manner to open-source software development, TINI has been subject to the input of a large community. This has enhanced the end product in a way that is not possible within a closed environment. Perhaps this open development is the most unusual aspect of the project.



TINI – for around $50, you get 10baseT ethernet, Dallas one-wire, I²C and RS232 interfaces, a processor, 512Kbyte of flash memory, 512Kbyte of NVRAM and an RTC. At its core is a soft Java virtual machine that's easy to update.

Getting started with TINI

In order to take full advantage of TINI's network abilities – it is the tiny Internet interface after all – you will need to connect TINI to a LAN. Many home users won't have their own IP network but this need not put you off. The box 'Establishing a network' shows how simple and cheap it is now to 'get wired'.

As has become traditional in the world of computing, our first TINI program is a network server version of the 'Hello World' classic. Groans aside, this simple application serves a number of purposes.

Firstly, the procedure to compile, build and load an application onto TINI is somewhat different from that of a normal Java application. A simple application can help with establishing that all is working correctly before we attempt something more extravagant.

Secondly, the program demonstrates a standard way of writing a multi-threaded network server. Next time, we'll be looking at something far more interesting but until then...

Listing 1 is the source code for our Hello Server. As mentioned, this program simply waits for a network connection, says Hello to the connecting machine, then closes the connection. Most Internet services such as web servers, etc., operate in a similar way.

Back to the example. Since this program uses Java's network and input/output libraries, these are imported at the top of the file. Next, define the name of the application – i.e. HelloServer. The main method forms the entry point to the program; this method is called by the

Establishing a network

While it is not actually necessary to network TINI in order to start experimenting, you'll be missing out on the whole idea that makes TINI so special – an embedded network node that runs Java.

Networking hardware has reduced in price dramatically in recent years. Browsing any of the monthly PC magazines shows a whole host of network cards for less than £15.00 and small 'micro' hubs – i.e. those with somewhere between 3 and 10 ports – for less than £30.00.

While it is beyond the scope of this article to delve into the intricacies of network engineering and the selection of networking hardware, we can examine the steps required to establish a simple TCP/IP network at home.

Of course you'll need a network card for your PC. I use a £10 NE2000 clone. Any 10baseT card should do, as long as it's supported by your chosen operating system.

You have two basic hardware options for connecting TINI to your personal computer. You can either use a crossover network cable or buy a small network hub and patch leads.* The crossover network cable has the advantage of being the cheapest but a small network hub offers far greater flexibility for not much more outlay.

Setting up your network

First, you'll need to ensure that you have TCP/IP networking installed for your chosen development platform – but this is the easy part as most systems come pre-configured with this option. If you dial into the Internet, you almost certainly have TCP/IP installed. If not, for a Windows platform, you can add this option in Control Panel -> Network -> Protocols.

Once you have all of the hardware installed and connected, you will need to configure each of the machines on the network with an address. If you use your PC for surfing the net then you should assign a network address from the range 192.168.0.0 to 192.168.255.255. These are from a range of special addresses that, if they 'escape' from your private network onto the Internet, can't cause any damage.

Addresses ending in '.0', '.1' and '.255' should not be used as they have special meaning on the network. For the rest of this article, I'll assume that you've chosen 192.168.0.50 for your PC and 192.168.0.100 for TINI

Addresses are assigned to PCs in Control Panel -> Network. Find the TCP/IP protocol that maps to your network card – not to your dial-up network adapter! Select properties and then the IP address tab. Enter your chosen IP address in the box provided and 255.255.255.0 into the 'subnet mask'. You can leave all of the other boxes empty. You can check your settings by using the commands winipcfg on windows 9x and ipconfig on NT.

TINI's IP address is set using the ipconfig command thus:

```
TINI /> ipconfig -a
192.168.0.100
```

You will need to login to your TINI board using JavaKit over a serial line in order to run ipconfig. Of course, you should have installed the firmware and booted TINI first!

Once you have two devices configured on the network, try pinging each one in turn from the other, e.g.,

```
C:\> ping 192.168.0.100
Pinging 192.168.0.100 with 32
bytes of data
Reply from 192.168.0.100:
bytes=32 time<10ms TTL=64

TINI /> ping 192.168.0.50
Got a reply from node
192.168.0.50/192.168.0.50
Sent 1 request(s), got 1
reply(s)
```

This shows that all's OK between the PC and TINI and *vice versa*.

*The author means a UTP crossover cable. Note that there are NE2000 clones that only have BNC connectors. Ed.

's network
:e after all – you
Iany home users
nis need not put
ɔrk' shows how
:d'.
ld of computing,
rver version of the
nis simple applica-

ild and load an
ferent from that of
plication can help
rrectly before we

a standard way of
er. Next time,
e interesting but

Hello Server. As
for a network
ng machine, then
ervices such as
way.
gram uses Java's
ese are imported at
me of the applica-
nod forms the
d is called by the

of the r
eck yo
mands
and ipconfig

using the

your TINI
a serial line in
course, you
firmware and

vices config-
pinging each
r, e.g.,
100
0 with 32

.100:
TL=64

.0.50
e
.0.50
ot 1

between the
rsa.

crossover cable.
) clones that only

Java virtual machine when it starts our program.

Our main method defines a `Socket` field called `client`, which is used for incoming connections. However, in order to receive these connect requests from clients, we have to use a '`ServerSocket`' to manage the process.

A `ServerSocket` 'binds' to a particular 'port' and listens for connections. When a `ServerSocket` accepts a connect request from a distant client, it passes the connection on, in the form of a `Socket` object, and goes back to listening. In this way, you don't have to wait for a client to finish using the Server program before others can connect.

It's a bit like phoning your bank's call centre. You dial a single number and your call is routed through to any available operator, allowing more calls to come in.

In order to be able to process multiple connections simultaneously, you can take advantage of Java 'threads' and make our server multithreaded. This is what's happening in the statement new `HelloServer(client)`. Our main method takes the `Socket` returned by the `ServerSocket` and creates a new `HelloServer` object to handle the connection. This object automatically starts a new thread upon creation and starts talking to the client.

This action can be seen in the constructor methods,

```
serverThread = new
Thread(this);
serverThread.start();
```

method calls.

The `start()` method eventually calls `run()` method. The run method first asks the `Socket` for something to write to – the `OutputStream` – and then turns this into something that can be printed to. You then simply print a message to this `Writer`.

Once we've sent our message we wait for a second, to allow you to read the message and then, rather rudely, we close our output channel and the socket, thus cutting off the client before they can respond to our `Hello`.

Building the application

Once you have entered the `HelloServer` code, you will need to turn it into a format suitable for the TINI.

First, compile the `HelloServer.java` file using,

```
javac -bootclasspath
<TINIPATH>/tiniclasses.jar
HelloServer.java
```

replacing the *<TINIPATH>* tag with the location of your TINI installation, for example, `C:\tini`.

The `-bootclasspath` directive allows the compiler, which is written in Java, to use a different set of core classes (java.lang, java.io, java.net, etc.,) from those compiled into the application. You won't be running under the standard JVM

So what can I do with it?

Interestingly, Tina's manufacturer, Dallas Semiconductor, is still a technology driven company. From the outsider's view – and this is often reflected in Dallas engineers' posts to the TINI mail list – it seems as if the company produces numerous clever solutions just waiting for a problem to come along.

TINI is more than just a rather cool toy though; it is a near-complete implementation of the J2ME platform albeit in pre-production, beta form.

At present, judging from the 'TINI-users' list, real-world TINI applications range from data loggers, security systems and network server monitors to GPS-aware systems and simple dial-up gateways. TINI could be applied in almost any scenario requiring a networked controller; from remote surveillance with an off-the-shelf webcam and wireless LAN technology to home automation using a DS1920 and Tesco Direct to ensure that you always have a good supply of cold beer!

Listing 1. HelloServer.java. This is a simple server that says 'Hello' to any clients.

```
/* HelloServer.java
*/
import java.io.*;
import java.net.*;
public class HelloServer implements Runnable {
    public static void main(String args[]) {
        Socket client;
        try{
            //Create a new server listening on port 1234
            ServerSocket server = new ServerSocket(1234);
            while(true){
                //wait for a call from a client
                client = server.accept();

                //start a new HelloServer for this client
                new HelloServer(client);

            }
        }catch(Exception e) {
            //No error handling - it's only an example ;-)
        }
    }


    Thread serverThread;
    Socket sock;

    public HelloServer(Socket sock) {
        this.sock = sock;
        serverThread = new Thread(this);
        serverThread.start();
    }

    public void run() {
        try {
            //Say hello, wait a bit and then disconnect
                PrintWriter out =
                              new PrintWriter(sock.getOutputStream());

            out.println("Hello from Tini!");
            Thread.currentThread().sleep(1000);
            out.close();
            sock.close();
        }catch(Exception e) {
            //No error handling - it's only an example ;-)
        }
    }

}
```

remember; you'll be using the special TINI VM so special core libraries are needed.

This should produce a file called `HelloServer.class`. Now we need to convert this class file into a .tini file,

```
java -classpath <TINIPATH>\tini.jar
TINIConvertor -f HelloServer.class -o
HelloServer.tini -d
<TINIPATH>\firmware\tini.db
```

again replacing the *<TINIPATH>* tag with the location of your TINI installation.

This command should produce a HelloServer.tini file. You will now need to FTP this class onto your TINI board. Windows and Linux both include command-line FTP clients, or you might like to use something like CuteFTP.

Nearly there! Telnet to your TINI board:

```
C:\>telnet 192.168.0.100
```

(use the standard username root and the password tini)

After logging in, start the HelloServer with the command

```
java HelloServer.tini &
```

Now open another telnet window and connect to your TINI on port 1234:

```
C:\>telnet 192.168.0.100 1234
```

TINI should say Hello and then, after a second or so, disconnect you.

Until next time

That's all for now. Next time, I'll be looking at some of the more useful features of TINI, including the various web-enabling technologies available. I will be showing how to hook up the one-wire iButton interface to a web-server, an RS232 terminal and an i/o port to create a simple web-enabled security system. ■

Resources

http://www.ibutton.com/TINI	TINI homepage – hardware, firmware, mail lists etc.
http://www3.sympatico.ca/guillaume.fournier/	Excellent 'Getting Started' resource
http://java.sun.com/	Source of all things Java inc. JDK, javax.comm. required for use with TINI
http://www.apms.com.au/tini/	Another good TINI resource site